# Swarm of Drones Simulation Testbed – Part 1

## Abstract

The autonomous swarm of drones is a hot topic nowadays. The aim of this project is to provide an experimental platform for conducting testing on swarm of drones in a simulated environment. The drone’s flight is modelled based on boids algorithm using Repast Simphony as agent-based modeling toolkit. The drones have cameras being able to identify and engage targets in their area of responsibility. In order to have a more realistic visual representation of simulated world, Three.js is employed to display 3D animated objects. The project is available on GitHub as drone\_swarm (https://github.com/traiannicula/drone\_swarm) public repository.

## Motivation

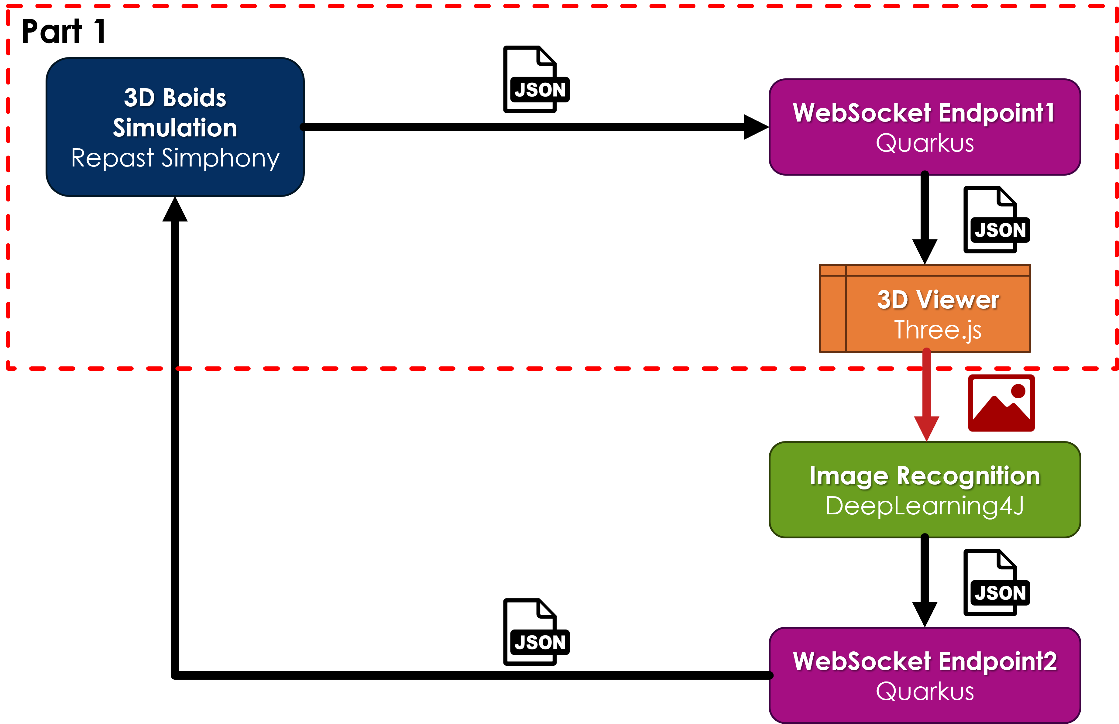
The idea of this project is not new, I have been thinking about it for few years now. The 2019 movie “Angel Has Fallen” was also an important incentive to proceed starting this work.

Coming from a military career I would like to further contribute to this domain proving once again that general purpose modeling and simulation (M&S) is a great tool for concept development and experimentation (CD&E). To be honest I do not have the means nor the technical knowledge to build a real swarm of drones, but this is possible in a simulation environment running on my desktop. Although the simulation is a representation of reality it can still provide useful insights into operational value of swarm of drones for combat or surveillance and reconnaissance.

On the other hand, now, I teach two courses at Military Technical Academy of Bucharest. One is a general-purpose M&S course and the other is Programming Paradigms in Java. With this project I want to persuade my students that both topics are worth learning.

## Project design

The ultimate goal of the project is to implement architectural diagram displayed below. I do not have at this point a full implementation but a partial one as marked on diagram as Part 1.



There is the simulation component developed as Repast Simphony (https://repast.github.io/) project. It implements the model and runs the simulation. It is also responsible of data exchange, in JSON format via WebSocket, with the visualization component based on Three.js (https://threejs.org/).

The JSON data are vectors with locations and orientations of drone agents.

There is no complex interaction usually encountered in 3D computed games like collision detection on Three.js side. The viewer component just mountain 3D objects in synchronization with the simulation. Each drone object has its own camera which are going to be used later for target identification by image recognition and classification. Image recognition and classification is not implemented in Part 1.

Quarkus holds WebSocket endpoints but it also acts as HTTP server for Three.js application. In future development it may also integrate DeepLearning4J as Java solution for image classification.

Once a target is classified the simulation will be notified by JSON data sent via WebSocket. The simulation shall conduct a target acquisition process and shall assign suicide mission to certain drones. In Part 1 there is a target acquisition but it doesn’t take into account what the drones “see”.

## The Model

The model is based on Flock project shipped as example with Repast Simphony 2.11.0 distribution. Flocking model simulates a large flock of prey birds that attempt to avoid predator birds. The birds fly in a 3D box following the 3 rules of the boids: separation, alignment, and cohesion. The behavior at box borders can be wrap around (torus), bouncy (reflection) or infinite (no border).

In my model I picked infinite borders as being more realistic than the other two. The prey agents are the swarm of drones and their status is sent to Three.js viewer. The predator agents exist only in simulation with the purpose of stirring up the drones. I soon discovered that flock like behavior combined with infinite borders lead to drones drifting away from box. To keep the swarm confined into the box I added an attractor agent.

The model has also target agents that are clamped to ground and move with maximum speed on rectilinear trajectory.

In Part 1, it is supposed that all drones “see” the targets which obvious is not a realistic scenario. This will be changed and target acquisition will use drone cameras as sensors and image recognition and classification software to identify the targets.

In the current implementation drones are assigned to targets based on proximity. When the number of drones on a certain target reaches a threshold, the target manager agent converts drones from prey agents to killer agents with the mission to get to target as fast as possible. The prey and killer agents have the same visual representation in Three.js viewer although their behavior in simulation is totally different.

There is also a scene updated agent responsible of sending locations and angles for drones and targets via WebSocket to Three.js viewer. On the simulation side I used Java-WebSocket library (https://github.com/TooTallNate/Java-WebSocket)

The model is highly configurable by 22 parameters that control the number of agents, maximum speed, acceleration, repel and attraction forces, different limit distances.

## Challenges

The complex dynamics of drome’s flight generates a lot of noise especially on drone’s velocity. The velocity is used to calculate pitch and yaw angles. For now, roll angle is ignored being set to zero.

When I initially sent data to Three.js viewer I got a lot of jitters. The solution I came up was to do velocity smoothing on simulation side. I chose moving average interpolation method and I got significant better visual results. Encouraged by the good outcome I used same technique for drone’s locations.

## Way Ahead

Capitalizing on good results obtained in Part 1 I would like to continue implementing the rest of the architectural diagram without setting a timeframe.

The first issue I had to tackle is how render images from perspective cameras attached to drones. Obviously, it has to be done on Three.js side. Doing some initial research, THREE.WebGLRenderTarget, in conjunction with a hidden canvas, seems to be the way to go. I do not know anything about performance so I have to do benchmark tests. I expect to not be able to render scene images from all drone’s cameras. Most likely I have to establish a reasonable number of scouts that will be used to generate images for further processing.

Assuming I will be able to overcome image generation the next step will be image recognition and classification. I will like to use DeepLearning4J just to have a Java and JavaScript solution.

The image recognition and classification shall generate usable JSON data which is sent back to simulation for target acquisition and mission assignment for drones.

The target manager agent shall process JSON data received, identify targets and assign drones for each probably using proximity criteria.

I have no plans to implement a solution like this on real life devices. As I said before I lack the technical knowledge for it.

## Acknowledgements

This endeavor would not have been possible without the help of my son Constantin Nicula. He provided useful insights and suggestions especially on computer gaming techniques and design also. He is a GitHub contributor at <https://github.com/ConstantinNicula>.

The 3D objects used in this project were downloaded from sketchfab.com (https://sketchfab.com/feed)